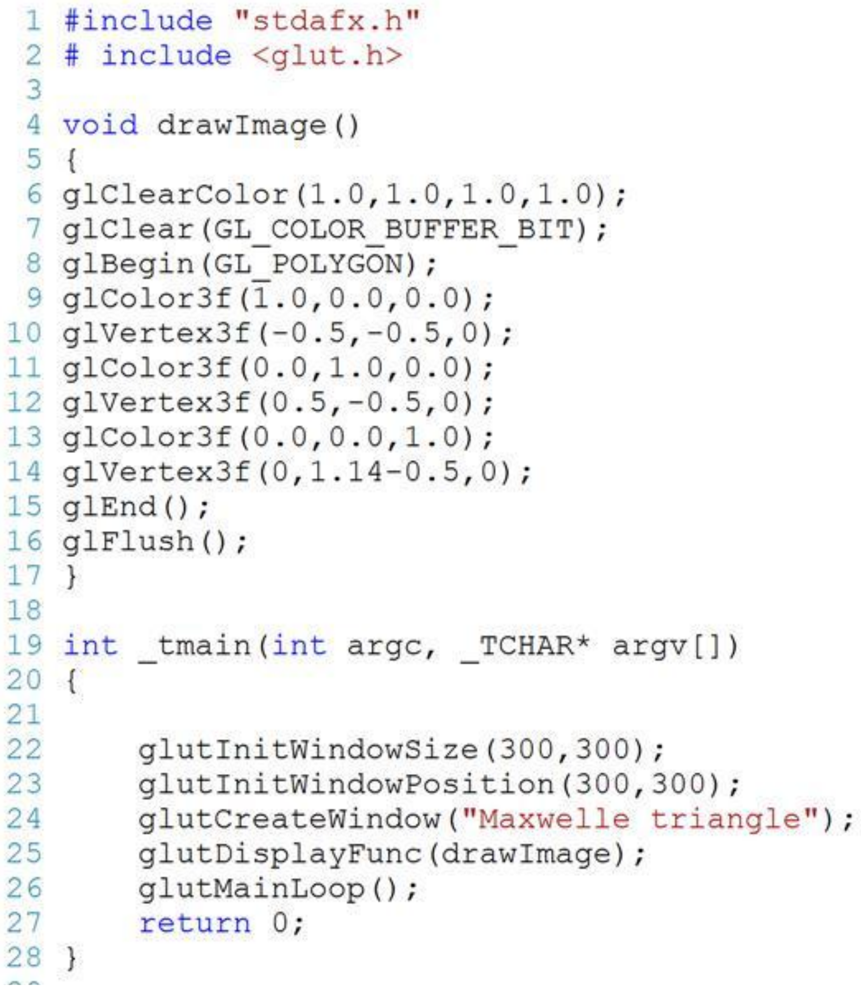
Sheet 4 solution

1. a) The program displays a circle instead of a sphere. The reason for this is viewing setting. The program does not set any viewing setting. Hence, the viewing is set to the default in OpenGL which makes the viewing volume as a cube 2x2x2 centered at the origin. The contents of this cube are projected on the image plane using orthogonal projection. Since our sphere is centered at the origin, it appears as a circle instead of a sphere.

   b) By changing the angle steps, we control the approximation. Smaller angle steps should give smother approximation. The reverse is also true.

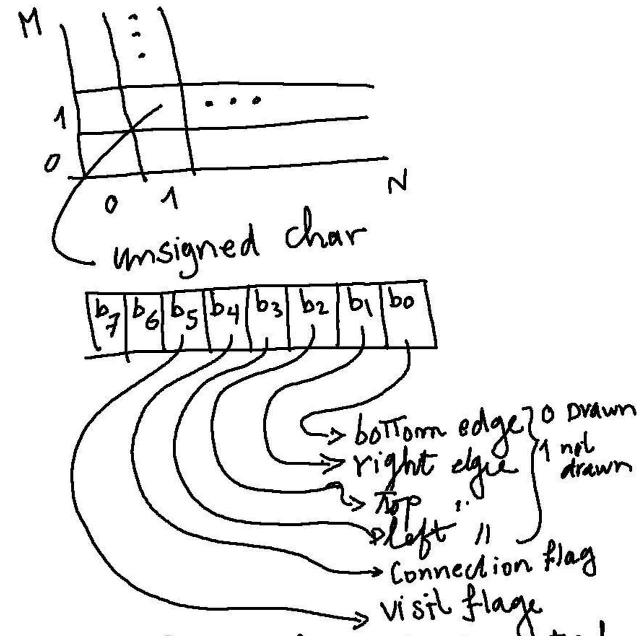
2. The following equations is used to convert from CMY subtractive color model to the RGB mode R=1.0-C, G=1.0-M, B=1-Y.

3.



```
#include "stdafx.h"
# include <glut.h>

void drawImage()
{
glClearColor(1.0,1.0,1.0,1.0);
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POLYGON);
glColor3f(1.0,0.0,0.0);
glVertex3f(-0.5,-0.5,0);
glColor3f(0.0,1.0,0.0);
glVertex3f(0.5,-0.5,0);
glColor3f(0.0,0.0,1.0);
glVertex3f(0,1.14-0.5,0);
glEnd();
glFlush();
}

int _tmain(int argc, _TCHAR* argv[])
{

    glutInitWindowSize(300,300);
    glutInitWindowPosition(300,300);
    glutCreateWindow("Maxwelle triangle");
    glutDisplayFunc(drawImage);
    glutMainLoop();
    return 0;
}
```

4.

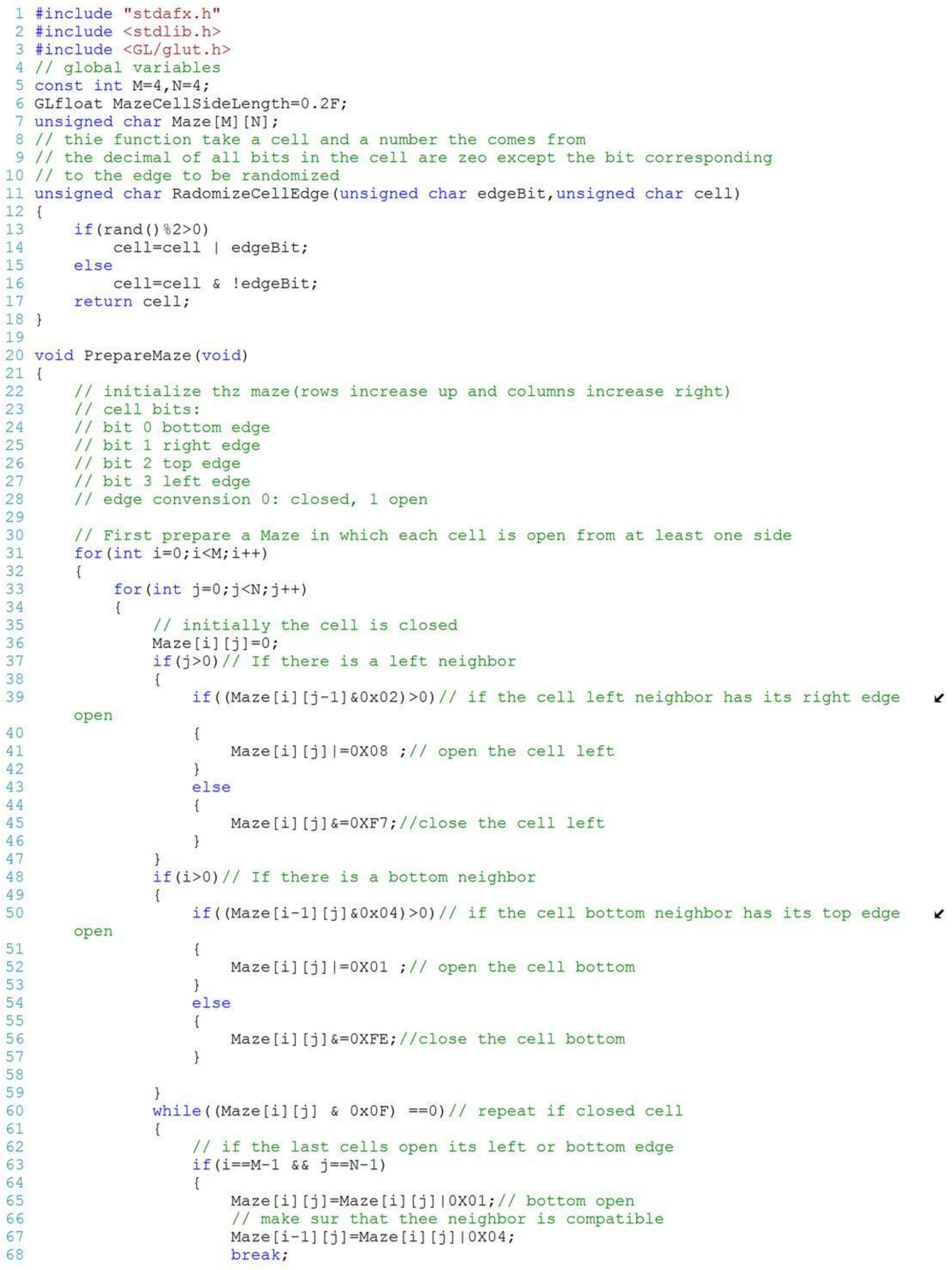


Connection flag: 1 → the cell is connected to every other cell through open (not drawn) edgos

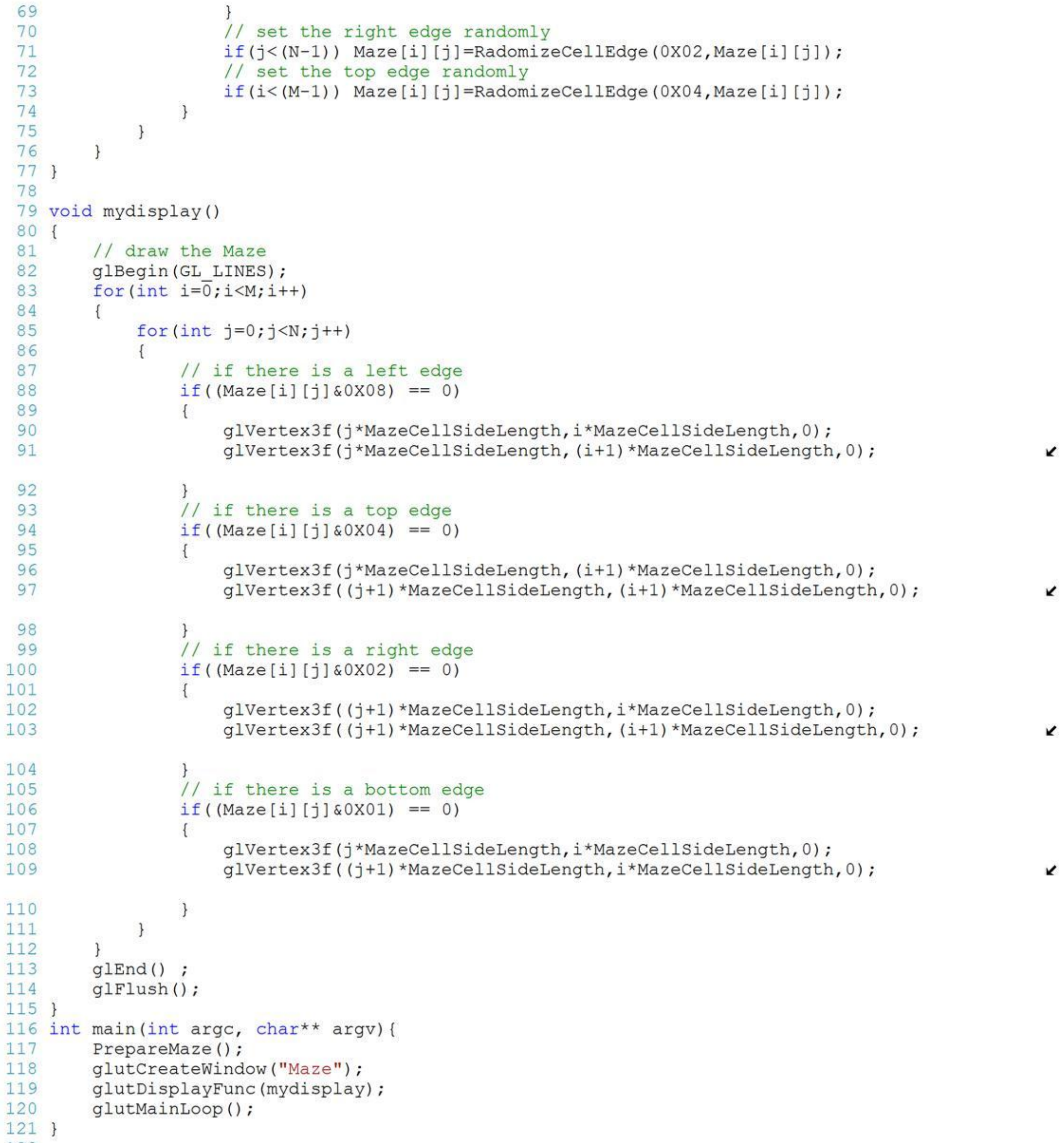
visit flage: 1 → the Cell is visited in the current connectivity check

Drawing the grid

we start with the Cell (0,0) which must have a left and bottom edgos closed because they are on the boundrios. then we choos randomly if the right and bottom edge are open or not. if a cell right or bottom is on the boundry the mide closed, otherwise it is set open or closed randomly

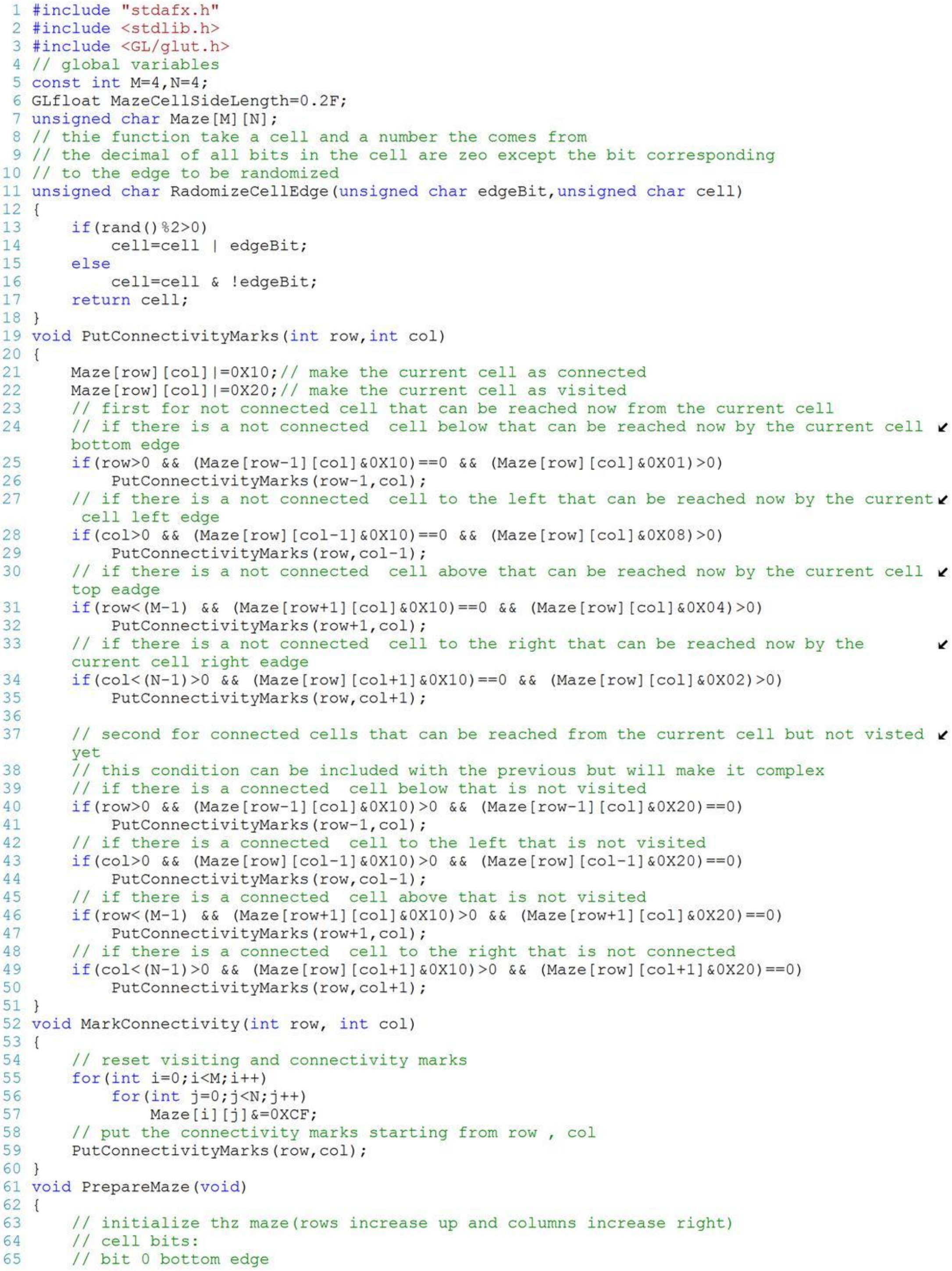


```
#include "stdafx.h"
#include <stdlib.h>
#include <GL/glut.h>
// global variables
const int M=4,N=4;
GLfloat MazeCellSideLength=0.2F;
unsigned char Maze[M][N];
// thie function take a cell and a number the comes from
// the decimal of all bits in the cell are zeo except the bit corresponding
// to the edge to be randomized
unsigned char RadomizeCellEdge(unsigned char edgeBit,unsigned char cell)
{
    if(rand()%2>0)
        cell=cell | edgeBit;
    else
        cell=cell & !edgeBit;
    return cell;
}

void PrepareMaze(void)
{
    // initialize thz maze(rows increase up and columns increase right)
    // cell bits:
    // bit 0 bottom edge
    // bit 1 right edge
    // bit 2 top edge
    // bit 3 left edge
    // edge conversion 0: closed, 1 open

    // First prepare a Maze in which each cell is open from at least one side
    for(int i=0;i<M;i++)
    {
        for(int j=0;j<N;j++)
        {
            // initially the cell is closed
            Maze[i][j]=0;
            if(j>0)// If there is a left neighbor
            {
                if((Maze[i][j-1]&0x02)>0)// if the cell left neighbor has its right edge open
                {
                    Maze[i][j]|=0X08 ;// open the cell left
                }
                else
                {
                    Maze[i][j]&=0XF7;//close the cell left
                }
            }
            if(i>0)// If there is a bottom neighbor
            {
                if((Maze[i-1][j]&0x04)>0)// if the cell bottom neighbor has its top edge open
                {
                    Maze[i][j]|=0X01 ;// open the cell bottom
                }
                else
                {
                    Maze[i][j]&=0XFE;//close the cell bottom
                }

            }
            while((Maze[i][j] & 0x0F) ==0)// repeat if closed cell
            {
                // if the last cells open its left or bottom edge
                if(i==M-1 && j==N-1)
                {
                    Maze[i][j]=Maze[i][j]|0X01;// bottom open
                    // make sur that thee neighbor is compatible
                    Maze[i-1][j]=Maze[i][j]|0X04;
                    break;
```

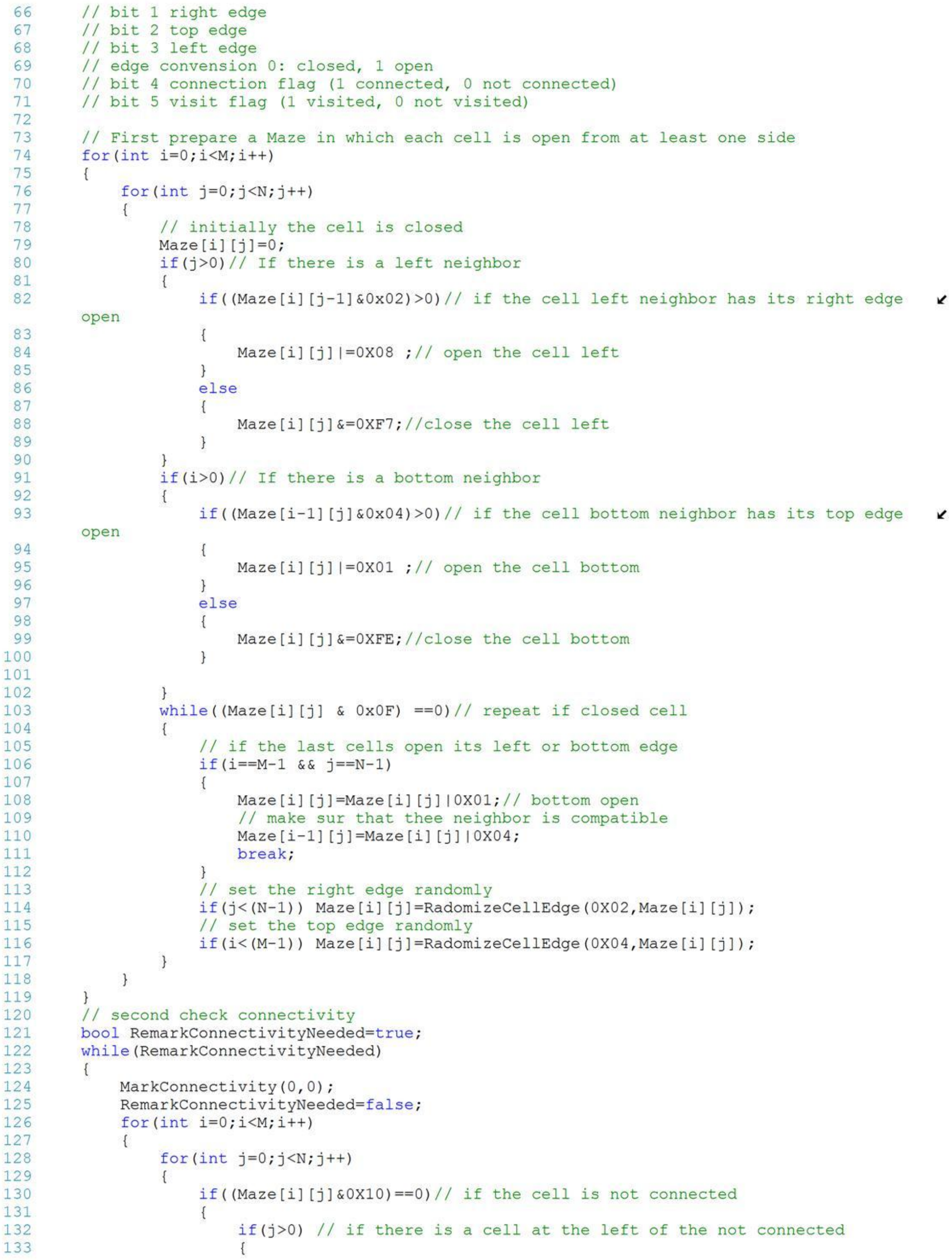


```
                }
                // set the right edge randomly
                if(j<(N-1)) Maze[i][j]=RadomizeCellEdge(0X02,Maze[i][j]);
                // set the top edge randomly
                if(i<(M-1)) Maze[i][j]=RadomizeCellEdge(0X04,Maze[i][j]);
            }
        }
    }
}

void mydisplay()
{
    // draw the Maze
    glBegin(GL_LINES);
    for(int i=0;i<M;i++)
    {
        for(int j=0;j<N;j++)
        {
            // if there is a left edge
            if((Maze[i][j]&0X08) == 0)
            {
                glVertex3f(j*MazeCellSideLength,i*MazeCellSideLength,0);
                glVertex3f(j*MazeCellSideLength,(i+1)*MazeCellSideLength,0);

            }
            // if there is a top edge
            if((Maze[i][j]&0X04) == 0)
            {
                glVertex3f(j*MazeCellSideLength,(i+1)*MazeCellSideLength,0);
                glVertex3f((j+1)*MazeCellSideLength,(i+1)*MazeCellSideLength,0);

            }
            // if there is a right edge
            if((Maze[i][j]&0X02) == 0)
            {
                glVertex3f((j+1)*MazeCellSideLength,i*MazeCellSideLength,0);
                glVertex3f((j+1)*MazeCellSideLength,(i+1)*MazeCellSideLength,0);

            }
            // if there is a bottom edge
            if((Maze[i][j]&0X01) == 0)
            {
                glVertex3f(j*MazeCellSideLength,i*MazeCellSideLength,0);
                glVertex3f((j+1)*MazeCellSideLength,i*MazeCellSideLength,0);

            }
        }
    }
    glEnd() ;
    glFlush();
}
int main(int argc, char** argv){
    PrepareMaze();
    glutCreateWindow("Maze");
    glutDisplayFunc(mydisplay);
    glutMainLoop();
}
```

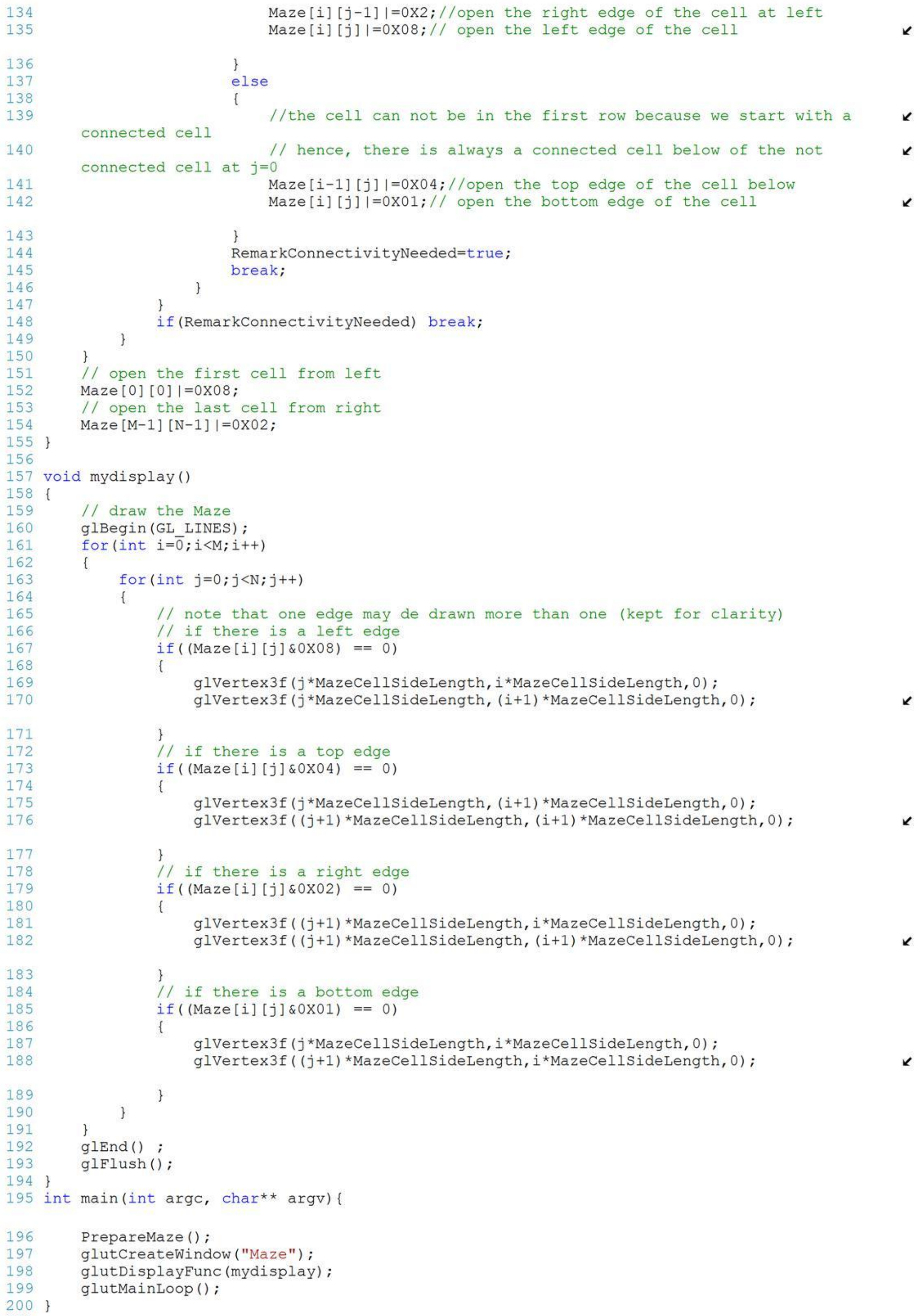
5.



```
#include "stdafx.h"
#include <stdlib.h>
#include <GL/glut.h>
// global variables
const int M=4,N=4;
GLfloat MazeCellSideLength=0.2F;
unsigned char Maze[M][N];
// thie function take a cell and a number the comes from
// the decimal of all bits in the cell are zeo except the bit corresponding
// to the edge to be randomized
unsigned char RadomizeCellEdge(unsigned char edgeBit,unsigned char cell)
{
    if(rand()%2>0)
        cell=cell | edgeBit;
    else
        cell=cell & !edgeBit;
    return cell;
}
void PutConnectivityMarks(int row,int col)
{
    Maze[row][col]|=0X10;// make the current cell as connected
    Maze[row][col]|=0X20;// make the current cell as visited
    // first for not connected cell that can be reached now from the current cell
    // if there is a not connected  cell below that can be reached now by the current cell bottom edge
    if(row>0 && (Maze[row-1][col]&0X10)==0 && (Maze[row][col]&0X01)>0)
        PutConnectivityMarks(row-1,col);
    // if there is a not connected  cell to the left that can be reached now by the current cell left edge
    if(col>0 && (Maze[row][col-1]&0X10)==0 && (Maze[row][col]&0X08)>0)
        PutConnectivityMarks(row,col-1);
    // if there is a not connected  cell above that can be reached now by the current cell top eadge
    if(row<(M-1) && (Maze[row+1][col]&0X10)==0 && (Maze[row][col]&0X04)>0)
        PutConnectivityMarks(row+1,col);
    // if there is a not connected  cell to the right that can be reached now by the current cell right eadge
    if(col<(N-1)>0 && (Maze[row][col+1]&0X10)==0 && (Maze[row][col]&0X02)>0)
        PutConnectivityMarks(row,col+1);

    // second for connected cells that can be reached from the current cell but not visted yet
    // this condition can be included with the previous but will make it complex
    // if there is a connected  cell below that is not visited
    if(row>0 && (Maze[row-1][col]&0X10)>0 && (Maze[row-1][col]&0X20)==0)
        PutConnectivityMarks(row-1,col);
    // if there is a connected  cell to the left that is not visited
    if(col>0 && (Maze[row][col-1]&0X10)>0 && (Maze[row][col-1]&0X20)==0)
        PutConnectivityMarks(row,col-1);
    // if there is a connected  cell above that is not visited
    if(row<(M-1) && (Maze[row+1][col]&0X10)>0 && (Maze[row+1][col]&0X20)==0)
        PutConnectivityMarks(row+1,col);
    // if there is a connected  cell to the right that is not connected
    if(col<(N-1)>0 && (Maze[row][col+1]&0X10)>0 && (Maze[row][col+1]&0X20)==0)
        PutConnectivityMarks(row,col+1);
}
void MarkConnectivity(int row, int col)
{
    // reset visiting and connectivity marks
    for(int i=0;i<M;i++)
        for(int j=0;j<N;j++)
            Maze[i][j]&=0XCF;
    // put the connectivity marks starting from row , col
    PutConnectivityMarks(row,col);
}
void PrepareMaze(void)
{
    // initialize thz maze(rows increase up and columns increase right)
    // cell bits:
    // bit 0 bottom edge
```



```
// bit 1 right edge
// bit 2 top edge
// bit 3 left edge
// edge convension 0: closed, 1 open
// bit 4 connection flag (1 connected, 0 not connected)
// bit 5 visit flag (1 visited, 0 not visited)

// First prepare a Maze in which each cell is open from at least one side
for(int i=0;i<M;i++)
{
    for(int j=0;j<N;j++)
    {
        // initially the cell is closed
        Maze[i][j]=0;
        if(j>0)// If there is a left neighbor
        {
            if((Maze[i][j-1]&0x02)>0)// if the cell left neighbor has its right edge open
            {
                Maze[i][j]|=0X08 ;// open the cell left
            }
            else
            {
                Maze[i][j]&=0XF7;//close the cell left
            }
        }
        if(i>0)// If there is a bottom neighbor
        {
            if((Maze[i-1][j]&0x04)>0)// if the cell bottom neighbor has its top edge open
            {
                Maze[i][j]|=0X01 ;// open the cell bottom
            }
            else
            {
                Maze[i][j]&=0XFE;//close the cell bottom
            }

        }
        while((Maze[i][j] & 0x0F) ==0)// repeat if closed cell
        {
            // if the last cells open its left or bottom edge
            if(i==M-1 && j==N-1)
            {
                Maze[i][j]=Maze[i][j]|0X01;// bottom open
                // make sur that thee neighbor is compatible
                Maze[i-1][j]=Maze[i][j]|0X04;
                break;
            }
            // set the right edge randomly
            if(j<(N-1)) Maze[i][j]=RadomizeCellEdge(0X02,Maze[i][j]);
            // set the top edge randomly
            if(i<(M-1)) Maze[i][j]=RadomizeCellEdge(0X04,Maze[i][j]);
        }
    }
}
// second check connectivity
bool RemarkConnectivityNeeded=true;
while(RemarkConnectivityNeeded)
{
    MarkConnectivity(0,0);
    RemarkConnectivityNeeded=false;
    for(int i=0;i<M;i++)
    {
        for(int j=0;j<N;j++)
        {
            if((Maze[i][j]&0X10)==0)// if the cell is not connected
            {
                if(j>0) // if there is a cell at the left of the not connected
                {
```



```
                        Maze[i][j-1]|=0X2;//open the right edge of the cell at left
                        Maze[i][j]|=0X08;// open the left edge of the cell
                    }
                    else
                    {
                        //the cell can not be in the first row because we start with a connected cell
                        // hence, there is always a connected cell below of the not connected cell at j=0
                        Maze[i-1][j]|=0X04;//open the top edge of the cell below
                        Maze[i][j]|=0X01;// open the bottom edge of the cell
                    }
                    RemarkConnectivityNeeded=true;
                    break;
                }
            }
            if(RemarkConnectivityNeeded) break;
        }
    }
    // open the first cell from left
    Maze[0][0]|=0X08;
    // open the last cell from right
    Maze[M-1][N-1]|=0X02;
}

void mydisplay()
{
    // draw the Maze
    glBegin(GL_LINES);
    for(int i=0;i<M;i++)
    {
        for(int j=0;j<N;j++)
        {
            // note that one edge may de drawn more than one (kept for clarity)
            // if there is a left edge
            if((Maze[i][j]&0X08) == 0)
            {
                glVertex3f(j*MazeCellSideLength,i*MazeCellSideLength,0);
                glVertex3f(j*MazeCellSideLength,(i+1)*MazeCellSideLength,0);
            }
            // if there is a top edge
            if((Maze[i][j]&0X04) == 0)
            {
                glVertex3f(j*MazeCellSideLength,(i+1)*MazeCellSideLength,0);
                glVertex3f((j+1)*MazeCellSideLength,(i+1)*MazeCellSideLength,0);
            }
            // if there is a right edge
            if((Maze[i][j]&0X02) == 0)
            {
                glVertex3f((j+1)*MazeCellSideLength,i*MazeCellSideLength,0);
                glVertex3f((j+1)*MazeCellSideLength,(i+1)*MazeCellSideLength,0);
            }
            // if there is a bottom edge
            if((Maze[i][j]&0X01) == 0)
            {
                glVertex3f(j*MazeCellSideLength,i*MazeCellSideLength,0);
                glVertex3f((j+1)*MazeCellSideLength,i*MazeCellSideLength,0);
            }
        }
    }
    glEnd() ;
    glFlush();
}
int main(int argc, char** argv){
    PrepareMaze();
    glutCreateWindow("Maze");
    glutDisplayFunc(mydisplay);
    glutMainLoop();
}
```